**User Guide: Security Report Generation Script**

**for**

**pdf\_test\_API\_and\_Access\_report.py**

**1. Overview**

This script processes security-related data from JSON files and creates a detailed security report in a PDF format. The report covers:

- Metadata about the security report

- Specific site data (like host, port, etc.)

- Test API data, including alert summaries and details

- Access control data for different user roles

- An appendix section for detailed instances of alerts

- Another appendix section for long URIs

**2. Dependencies**

This script relies on the following external libraries:

- json: To read and parse JSON files.

- datetime: To work with date and time.

- reportlab: To generate PDF reports.

Make sure to install reportlab with pip: **pip install reportlab**

**3. Global Variables – Constants – Styling – Headers**

- PDF\_PATH: specifies the location and filename format of the output PDF. **This filename can be edited by the user. The date/time stamp can be either retained or removed.**

- styles: contains styles for different parts of the report. **Titles, font size, borders, and cell styles can be edited by the user to style any new json file data or layout to the PDF document.**

- Various headers and style commands: define the structure and appearance of the report. **Headers can be edited or changed to suit new Jason files. Document styling can be edited by the user. Header styling can be edited by the user.**

**4.** **Functions**

**4.1. load\_json\_data(filename)**

**This function reads and parses a given JSON file to extract the contained data. If the file is not found or there is an issue with the JSON structure/content, an error message is printed and the program terminates.**

- Loads data from a given JSON file.

- Parameter: The name of the JSON file.

- Returns: The parsed JSON data.

**- This function does not need editing for styling or document layout changes.**

**4.2. process\_uri(uri, long\_uri)**

**If the URI length exceeds 2000 characters, it's added to the 'long\_uri' list and a placeholder is returned indicating its location in the appendix. Otherwise, the URI itself is returned formatted.**

- Processes URIs, and if they are too long, they are referenced in an appendix.

- Parameters: URI string and a list to collect long URIs.

- Returns: The processed URI or a reference to the appendix.

**- This function can be edited to handle long URIs in a different manner. For example, the user may only desire the first 100 characters of a URI to be shown in the report. This can be changed here.**

**4.3. add\_metadata\_to\_elements(report\_data, elements)**

**This function takes in the report data dictionary which contains metadata details like program name, version, and generation time. It then prepares formatted metadata strings using these details and adds them as Paragraph objects to the given elements list.**

- Adds metadata about the report to the provided list.

- Parameters: The report data and a list of report elements.

**- This function has been customised to the report\_json file. It can be edited to suit any new style of json report.**

**4.4. add\_site\_data\_to\_elements(site, elements)**

**This function takes in the site data dictionary which contains details like site name, host, port, and SSL status. It then prepares formatted strings using these details and adds them as Paragraph objects to the given elements list. Additionally, the SSL status is interpreted as "Enabled" or "Disabled" based on its boolean value in the site data.**

- Adds site-specific data to the provided list.

- Parameters: The site data and a list of report elements.

**- This function has been customised to the report\_json file. It can be edited to suit any new style of json report.**

**4.5. process\_test\_api\_data(report\_data, elements)**

**This function iterates over the sites present in the report data, then processes each alert associated with the site. For each alert, a table is prepared which includes alert details and instance information. Styling is applied based on predefined styles. The function adds each constructed table to the elements list, and follows it with a page break for report separation. The 'Alert' column in the table has a special red background styling.**

- Processes test API data and adds it to the provided list.

- Parameters: The report data and a list of report elements.

**- This function has been customised to the report\_json file. It can be edited to suit any new style of json report.**

**4.6. process\_access\_data(access\_report, elements)**

**This function takes an access report detailing which roles have access to various endpoints and constructs a unified table that displays this information. The resulting table specifies whether a particular role has access to an endpoint with "Yes" or "No". Special styling is applied to the table, notably highlighting cells that indicate "No" access with a red background. This processed table is then appended to the provided elements list, which is typically used to generate a PDF report.**

- Processes access control data and adds it to the provided list.

- Parameters: The access report data and a list of report elements.

**- This function has been customised to the access\_test\_results.json file. It can be edited to suit any new style of json report.**

**4.7. process\_appendix\_for\_instances(report\_data, elements, long\_uri)**

**This function parses an access report that details which roles have access to various endpoints. It constructs a unified table to summarise the access privileges. The resulting table indicates whether each role has access to an endpoint with "Yes" or "No". The function also applies styling to the table rows, especially highlighting cells with "No" access using a red-orange background.**

- Processes alert instance data for the appendix section.

- Parameters: The report data, a list of report elements, and a list for collecting long URIs.

**- This function can be removed if Instances information is not required by the user.**

**4.8. process\_appendix\_for\_long\_uris(long\_uri, elements)**

**This function checks if there are any long URIs in the provided list. If so, it adds an appendix section to the 'elements' list, detailing each long URI with an associated entry number. Each URI is displayed using a specific style and separated with a spacer for readability.**

- Processes long URIs for the appendix section.

- Parameters: The list of long URIs and a list of report elements.

**- This function can be removed if the requirement to handle long URIs is not desired by the user.**

**4.9. create\_report()**

**This function loads data from JSON files, processes the data to organize it in a structured manner, and then generates a comprehensive security report in PDF format. The PDF is generated using ReportLab's SimpleDocTemplate and various helper functions.**

- The main function to generate the PDF report.

**- The user can enter additional report content here by following the code path of the report\_data and access\_report variables. To add new reports the user will also need to copy/mimic functions 4.5 and 4.6. The user would also need to add headers for any new json files added. The tables sizes for the new reports can be adjusted by copying/mimicking the table\_for\_alert variable data as shown at line 238 in the Python code. Further styling and cell changes can be adjusted as per the code in functions 4.5 and 4.6 when constructing a new table, from adding a new json file.**

**5. Running the Script**

To generate a security report:

1. Place the security report data in a file named **report\_json.**

2. Place the access testing results in a file named access\_testing/access\_test\_results.json.

3. Run the script: **python report\_generation\pdf\_test\_API\_and\_Access\_report.py**

**After successful execution, a security report in PDF format will be generated with the naming convention test\_api\_and\_access\_report\_<timestamp>.pdf.**

**6. Customisation**

- If you need to customise the appearance or structure of the report, modify the respective headers, styles, or style commands.

- If you need to process different data or include additional sections, add or modify the respective processing functions.

7. Troubleshooting

- Ensure JSON files are well-structured and available at the specified locations.

- Ensure the reportlab library is correctly installed.

- Check error messages for more information about any issues.